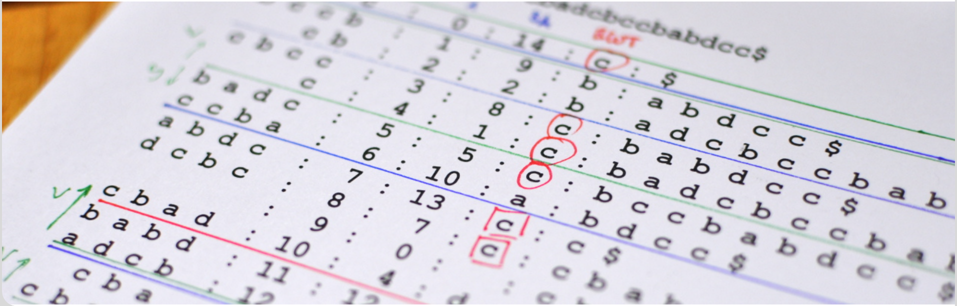
Karlsruhe Institute of Technology

Scalable Construction of Text Indexes with Thrill

Timo Bingmann, Simon Gog, and Florian Kurpicz · IEEE Big Data · December 12th, 2018

INSTITUTE OF THEORETICAL INFORMATICS – ALGORITHMICS



Example $T = [\overset{0}{\texttt{t}}\overset{1}{\texttt{o}}\overset{2}{\texttt{b}}\overset{3}{\texttt{e}}\overset{4}{\texttt{o}}\overset{5}{\texttt{r}}\overset{6}{\texttt{n}}\overset{7}{\texttt{o}}\overset{8}{\texttt{t}}\overset{9}{\texttt{t}}\overset{10}{\texttt{o}}\overset{11}{\texttt{b}}\overset{12}{\texttt{e}}\overset{13}{\texttt{\$}}]$

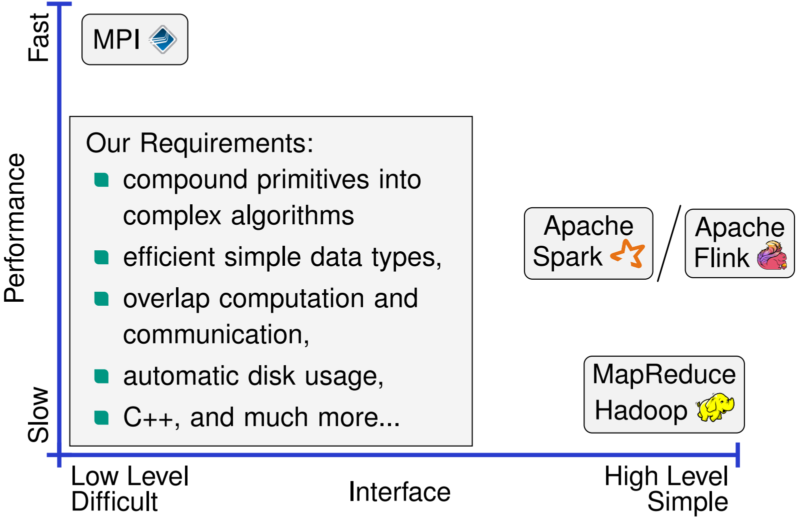
i	T_i
0	t o b e o r n o t t o b e $
1	o b e o r n o t t o b e $
2	b e o r n o t t o b e $
3	e o r n o t t o b e $
4	o r n o t t o b e $
5	r n o t t o b e $
6	n o t t o b e $
7	o t t o b e $
8	t t o b e $
9	t o b e $
10	o b e $
11	b e $
12	e $
13	$

Example $T = [\overset{0}{\mathtt{t}}\overset{1}{\mathtt{o}}\overset{2}{\mathtt{b}}\overset{3}{\mathtt{e}}\overset{4}{\mathtt{o}}\overset{5}{\mathtt{r}}\overset{6}{\mathtt{n}}\overset{7}{\mathtt{o}}\overset{8}{\mathtt{t}}\overset{9}{\mathtt{t}}\overset{10}{\mathtt{o}}\overset{11}{\mathtt{b}}\overset{12}{\mathtt{e}}\overset{13}{\mathtt{\$}}]$

SA_i	LCP_i	$T_{SA_i \ldots n}$
13	-	$
11	0	be$
2	2	beornottobe$
12	0	e$
3	1	eornottobe$
6	0	nottobe$
10	0	obe$
1	3	obeornottobe$
4	1	ornottobe$
7	1	ottobe$
5	0	rnottobe$
9	1	tobe$
0	4	tobeornottobe$
8	1	ttobe$

bwUniCluster
© KIT (SCC)

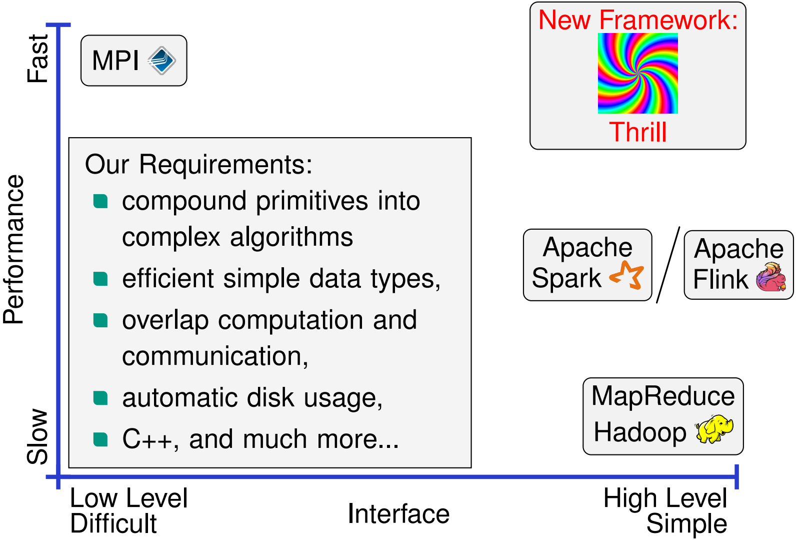
Big Data Batch Processing



Our Requirements:

- compound primitives into complex algorithms
- efficient simple data types,
- overlap computation and communication,
- automatic disk usage,
- C++, and much more...

Big Data Batch Processing



Our Requirements:

- compound primitives into complex algorithms
- efficient simple data types,
- overlap computation and communication,
- automatic disk usage,
- C++, and much more...

Thrill’s Goal and Current Status

An easy way to program fast distributed algorithms in C++.

Current Status:

- Open-source prototype at `http://github.com/thrill/thrill`.
- $\approx$ 60 K lines of C++14 code, written by $\geq$ 12 contributors.
- Published at IEEE Conference on Big Data [B, et al. ’16]
- Faster than Apache Spark and Apache Flink on five micro benchmarks: WordCount1000, WordCountCC, PageRank, TeraSort, and K-Means.

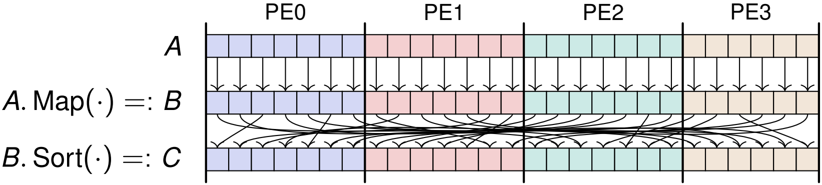
Case Studies:

- Five suffix sorting algorithms [B, Gog, Kurpicz, this presentation]
- Louvain graph clustering algorithm [Hamann et al. Euro-Par’18]
- Process scientific data on HPC (poster) [Karabin et al. SC’18]
- More examples: stochastic gradient descent, triangle counting, etc.
- Future: fault tolerance, scalability, and more applications.

Distributed Immutable Array (DIA)

User Programmer's View:

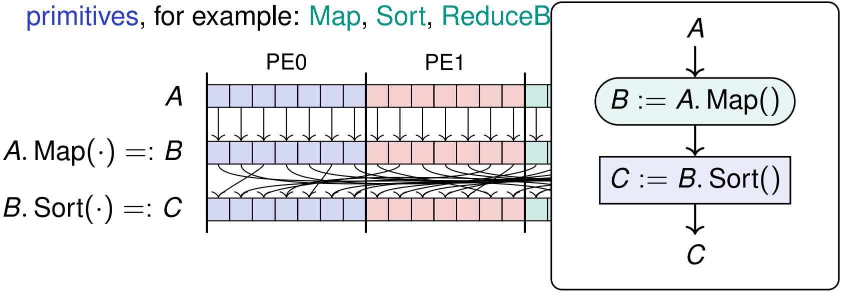
- `DIA<T>` = distributed array of items T on the cluster
- Cannot access items directly, instead use small set of scalable primitives, for example: Map, Sort, ReduceByKey, Zip, Window, etc.



Distributed Immutable Array (DIA)

User Programmer's View:

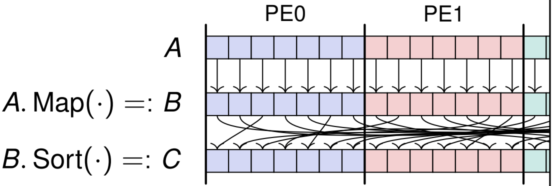
- `DIA<T>` = distributed array of items T on the cluster
- Cannot access items directly, instead use small set of scalable primitives, for example: Map, Sort, ReduceB

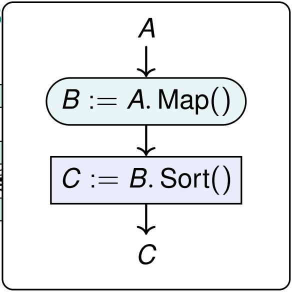


Distributed Immutable Array (DIA)

User Programmer's View:

- `DIA<T>` = distributed array of items T on the cluster
- Cannot access items directly, instead use small set of scalable primitives, for example: Map, Sort, ReduceB





Framework Designer's View:

- Goals: distribute work, optimize execution on cluster, add redundancy where applicable. ⇒ build data-flow graph.
- `DIA<T>` = pipelined chain of computations

Distributed Suffix Sorting

Prior Work on Distributed Algorithms:

- Early research with distributed sorting algorithms and using “pruned suffixes” and “sistrings” with additional characters. [KRRNZ, ICAPP’97]
- Emulation of sequential suffix sorting algorithms. [KN, SPIRE’99]
- With assumption that the entire text is on each processor. [FAK, ’01]

- First MPI implementation of DC3 using sample sort. [KS, PVM/MPI’06]

- Hadoop implementation of simple algorithms. [MBS, MapReduce’11]
- Implementation of [FAK01] named cloudSACA for Amazon Web Services. [AKA, MECBME’14]

- Implementation of prefix doubling using MPI. [FA, SC’15]

Paper: Scalable Suffix Sorting with Thrill

Implemented five algorithms for distributed external memory

- Prefix doubling using the inverse suffix array (DoublingW)
- Prefix doubling using sorting (DoublingS)
- Prefix doubling with discarding (Discarding)
- DC3
- DC7

Example $T = [\overset{0}{\mathtt{t}}\overset{1}{\mathtt{o}}\overset{2}{\mathtt{b}}\overset{3}{\mathtt{e}}\overset{4}{\mathtt{o}}\overset{5}{\mathtt{r}}\overset{6}{\mathtt{n}}\overset{7}{\mathtt{o}}\overset{8}{\mathtt{t}}\overset{9}{\mathtt{t}}\overset{10}{\mathtt{o}}\overset{11}{\mathtt{b}}\overset{12}{\mathtt{e}}\overset{13}{\mathtt{\$}}]$

i	T_i
0	t o b e o r n o t t o b e $
1	o b e o r n o t t o b e $
2	b e o r n o t t o b e $
3	e o r n o t t o b e $
4	o r n o t t o b e $
5	r n o t t o b e $
6	n o t t o b e $
7	o t t o b e $
8	t t o b e $
9	t o b e $
10	o b e $
11	b e $
12	e $
13	$

Example $T = [\overset{0}{\mathtt{t}}\overset{1}{\mathtt{o}}\overset{2}{\mathtt{b}}\overset{3}{\mathtt{e}}\overset{4}{\mathtt{o}}\overset{5}{\mathtt{r}}\overset{6}{\mathtt{n}}\overset{7}{\mathtt{o}}\overset{8}{\mathtt{t}}\overset{9}{\mathtt{t}}\overset{10}{\mathtt{o}}\overset{11}{\mathtt{b}}\overset{12}{\mathtt{e}}\overset{13}{\mathtt{\$}}]$

i		T_i	
13	0	$	
2	1	b	eornottobe$
11		b	e$
3	3	e	ornottobe$
12		e	$
6	5	n	ottobe$
1	6	o	beornottobe$
4		o	rnottobe$
7		o	ttobe$
10		o	be$
5	10	r	nottobe$
0	11	t	obeornottobe$
8		t	tobe$
9		t	obe$

Example $T = \overset{0}{\texttt{t}}\overset{1}{\texttt{o}}\overset{2}{\texttt{b}}\overset{3}{\texttt{e}}\overset{4}{\texttt{o}}\overset{5}{\texttt{r}}\overset{6}{\texttt{n}}\overset{7}{\texttt{o}}\overset{8}{\texttt{t}}\overset{9}{\texttt{t}}\overset{10}{\texttt{o}}\overset{11}{\texttt{b}}\overset{12}{\texttt{e}}\overset{13}{\texttt{\$}}]$

i		T_i			
13	0	$			
2	1	b	e	3	o r n o t t o b e $
11		b	e	3	$
3	3	e	o	6	r n o t t o b e $
12		e	$	0	
6	5	n	o		t t o b e $
1	6	o	b	1	e o r n o t t o b e $
4		o	r	10	n o t t o b e $
7		o	t	11	t o b e $
10		o	b	1	e $
5	10	r	n		o t t o b e $
0	11	t	o	6	b e o r n o t t o b e $
8		t	t	11	o b e $
9		t	o	6	b e $

Example $T = \overset{0}{\texttt{[t}}\overset{1}{\texttt{o}}\overset{2}{\texttt{b}}\overset{3}{\texttt{e}}\overset{4}{\texttt{o}}\overset{5}{\texttt{r}}\overset{6}{\texttt{n}}\overset{7}{\texttt{o}}\overset{8}{\texttt{t}}\overset{9}{\texttt{t}}\overset{10}{\texttt{o}}\overset{11}{\texttt{b}}\overset{12}{\texttt{e}}\overset{13}{\texttt{\$]}}$

i		T_i
13	0	$
2	1	b e 3 o r n o t t o b e $
11		b e 3 $
12	3	e $ 0
3		e o 6 r n o t t o b e $
6	5	n o t t o b e $
1	6	o b 1 e o r n o t t o b e $
10		o b 1 e $
4		o r 10 n o t t o b e $
7		o t 11 t o b e $
5	10	r n o t t o b e $
0	11	t o 6 b e o r n o t t o b e $
9		t o 6 b e $
8		t t 11 o b e $

0 1 2 3 4 5 6 7 8 9 10 11 12 13

Example *T* = [tobeornottobe$]

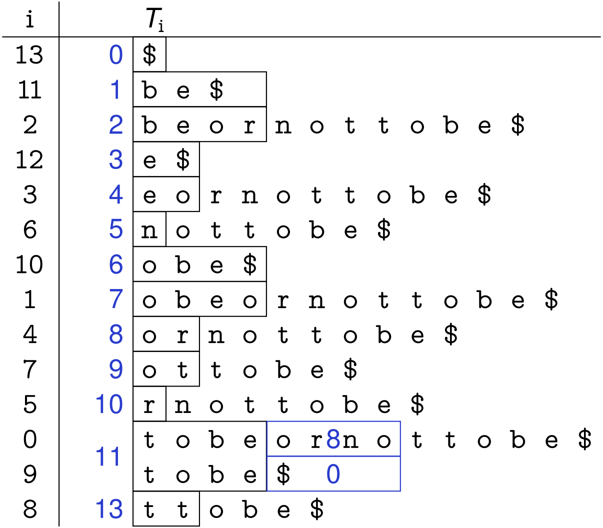
i		T_i
13	0	$
2	1	beo8rnottobe$
11		be$0
12	3	e$
3	4	eornottobe$
6	5	nottobe$
1	6	obe4ornottobe$
10		obe3$
4	8	ornottobe$
7	9	ottobe$
5	10	rnottobe$
0	11	tob1eornottobe$
9		tob1e$
8	13	ttobe$

0 1 2 3 4 5 6 7 8 9 10 11 12 13

Example $T = [\texttt{tobeornottobe\$}]$

i		T_i
13	0	$
11	1	b e $0
2	2	b e o8r n o t t o b e $
12	3	e $
3	4	e o r n o t t o b e $
6	5	n o t t o b e $
10	6	o b e $
1	7	o b e o r n o t t o b e $
4	8	o r n o t t o b e $
7	9	o t t o b e $
5	10	r n o t t o b e $
0	11	t o b1e o r n o t t o b e $
9		t o b1e $
8	13	t t o b e $

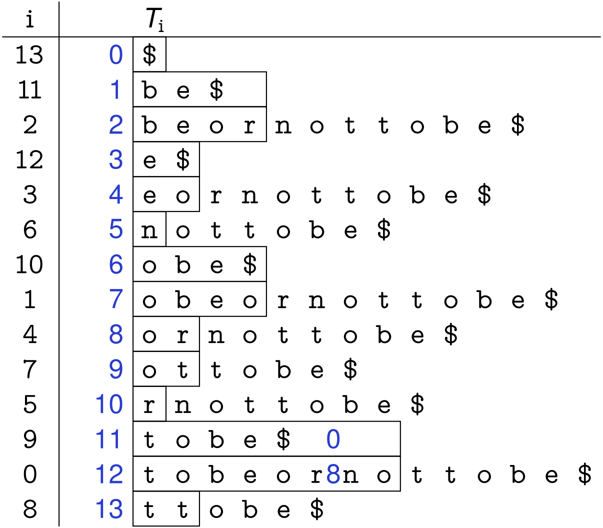
Example $T = [\overset{0}{\texttt{t}}\overset{1}{\texttt{o}}\overset{2}{\texttt{b}}\overset{3}{\texttt{e}}\overset{4}{\texttt{o}}\overset{5}{\texttt{r}}\overset{6}{\texttt{n}}\overset{7}{\texttt{o}}\overset{8}{\texttt{t}}\overset{9}{\texttt{t}}\overset{10}{\texttt{o}}\overset{11}{\texttt{b}}\overset{12}{\texttt{e}}\overset{13}{\texttt{\$}}]$



i		T_i
13	0	$
11	1	be$
2	2	beornottobe$
12	3	e$
3	4	eornottobe$
6	5	nottobe$
10	6	obe$
1	7	obeornottobe$
4	8	ornottobe$
7	9	ottobe$
5	10	rnottobe$
0	11	tobeor8nottobe$
9		tobe$ 0
8	13	ttobe$

0 1 2 3 4 5 6 7 8 9 10 11 12 13

Example $T = [\texttt{tobeornottobe\$}]$

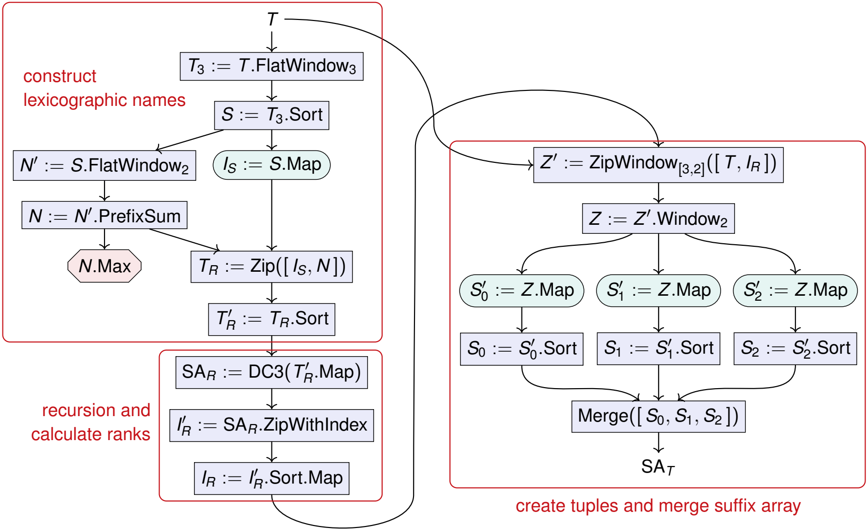


i		T_i
13	0	$
11	1	be$
2	2	beornottobe$
12	3	e$
3	4	eornottobe$
6	5	nottobe$
10	6	obe$
1	7	obeornottobe$
4	8	ornottobe$
7	9	ottobe$
5	10	rnottobe$
9	11	tobe$ 0
0	12	tobeor8nottobe$
8	13	ttobe$

Thrill Pseudo-Code for Prefix DoublingS

```
Function PrefixDoublingWithSorting($T \in \langle\Sigma\rangle$)
    $S := T$.Window$_2$($(i, [\,t_0, t_1\,]) \mapsto (i, t_0, t_1)$)        // Initial triples $(i, T[i], T[i+1])$.
    for $k := 1$ to $\lceil \log_2 |T| \rceil - 1$ do
        $S := S$.Sort($(i, n_0, n_1)$ by $(n_0, n_1)$)        // Sort triples by name pair.
        $N := S$.FlatWindow$_2$($(j, [\,a, b\,]) \mapsto$ CmpName(j, a, b))        // Outputs 0 or $j$.
        if $N$.Filter($(i, n) \mapsto (n = 0)$).Size() $= 1$ then        // If all names distinct, then
            return $N$.Map($(i, n) \mapsto i$)        // return names as suffix array, else
        $N := N$.PrefixSum($(i, n), (i', n') \mapsto (i', \max(n, n'))$)        // make new names.
        $N := N$.Sort($(i, n)$ by $i$)        // Compute ISA$^{2^k}$.
        $S := N$.Window$_{2^k+1}$($(j, [\,(i, n), \dots, (i', n')\,]) \mapsto$
            $\left(\begin{cases} (i, n, n') & \text{if } j + 2^k < |T|, \\ (i, n, 0) & \text{otherwise.} \end{cases}\right)$        // Compare names ISA$^{2^k}[i]$ and ISA$^{2^k}[i + 2^k]$.
```

Data-Flow Graph of DC3 with Recursion



construct lexicographic names

T

$T_3 := T.\text{FlatWindow}_3$

$S := T_3.\text{Sort}$

$N' := S.\text{FlatWindow}_2$

$I_S := S.\text{Map}$

$N := N'.\text{PrefixSum}$

$N.\text{Max}$

$T_R := \text{Zip}([\,I_S, N\,])$

$T'_R := T_R.\text{Sort}$

recursion and calculate ranks

$\text{SA}_R := \text{DC3}(T'_R.\text{Map})$

$I'_R := \text{SA}_R.\text{ZipWithIndex}$

$I_R := I'_R.\text{Sort.Map}$

$Z' := \text{ZipWindow}_{[3,2]}([\,T, I_R\,])$

$Z := Z'.\text{Window}_2$

$S'_0 := Z.\text{Map}$

$S'_1 := Z.\text{Map}$

$S'_2 := Z.\text{Map}$

$S_0 := S'_0.\text{Sort}$

$S_1 := S'_1.\text{Sort}$

$S_2 := S'_2.\text{Sort}$

$\text{Merge}([\,S_0, S_1, S_2\,])$

SA_T

create tuples and merge suffix array

Experiments on AWS EC2

Inputs: prefixes of

- **Wikipedia** XML dump (up to 125.6 GiB)
- **Gutenberg** text document corpus (up to 23 GiB)
- **Pi** digits of π ("3.1415926535...")

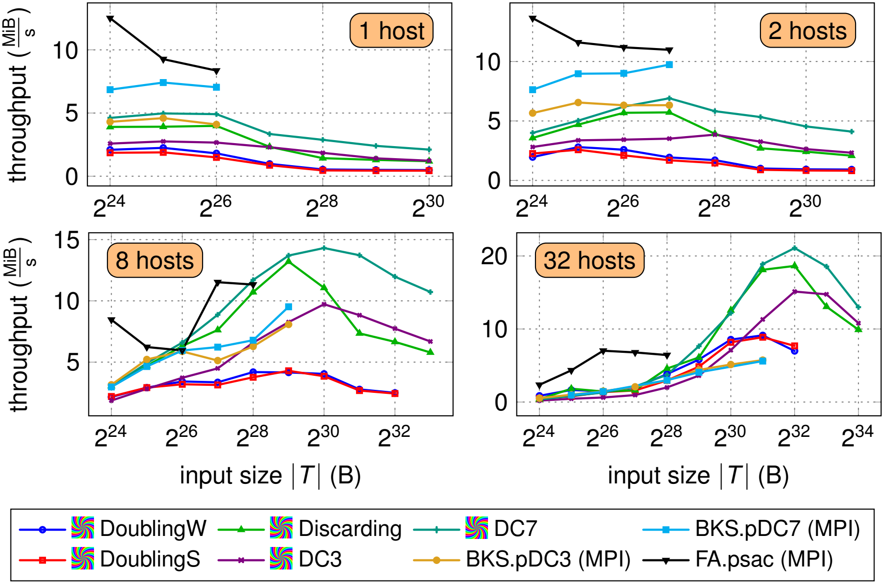
Machine:

- up to 32 × i3.4xlarge EC2 instances.
- 16-core Intel Xeon E5-2686 CPUs with 2.30 GHz
- 8 GB of RAM, and 2 × 1.9 TB NVMe SSDs

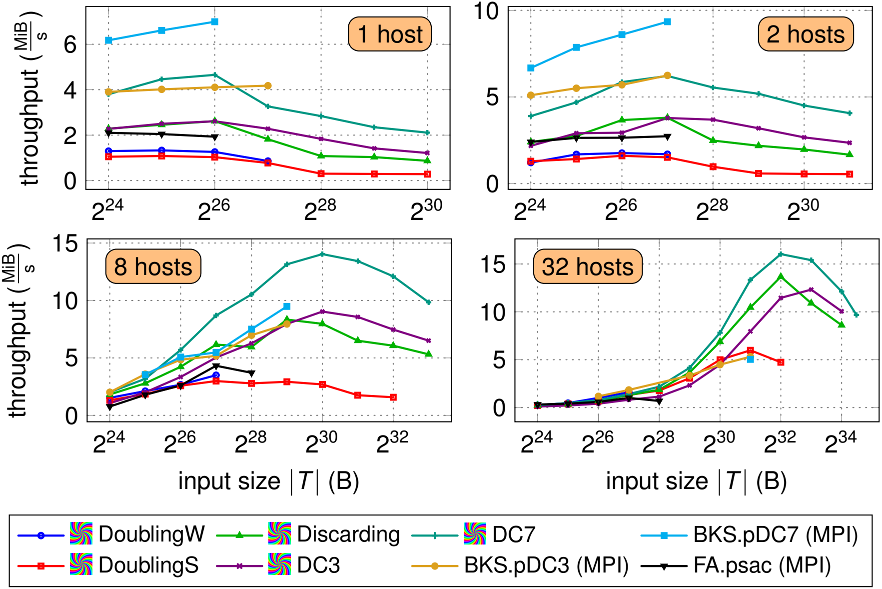
Competitors:

- BKS.pDC3 and BKS.pDC7 – MPI difference cover algorithm
- FA.psac – MPI prefix doubling algorithm
- also compared against fastest non-distributed algorithms:
  M.divsufsort, M.divsufsort.par, and M.sais.

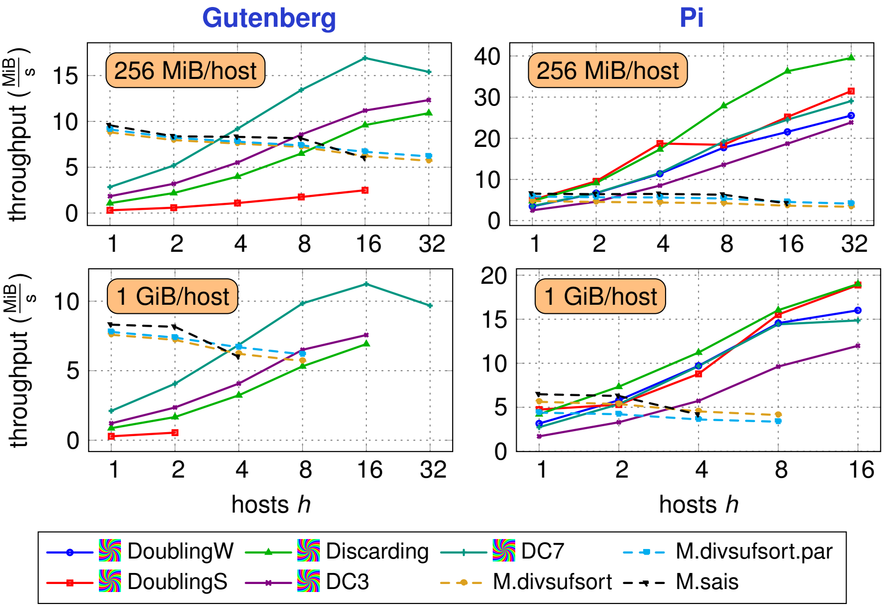
Suffix Sorting Wikipedia on AWS EC2



Suffix Sorting Gutenberg on AWS EC2

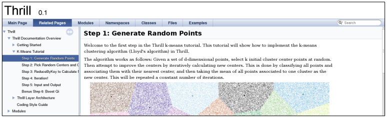


Weak Scaling and COST on AWS EC2



Current and Future Work

- Open-Source at `http://project-thrill.org` and Github.
- High quality, very modern C++14 code.
- A K-Mean tutorial is available!



Step 1: Generate Random Points

Welcome to the first step in the Thrill k-means tutorial. This tutorial will show how to implement the k-means clustering algorithm (Lloyd's algorithm) in Thrill.

The algorithm works as follows: Given a set of d-dimensional points, select k initial cluster center points at random. Then attempt to improve the centers by iteratively calculating new centers. This is done by classifying all points and associating them with their nearest center, and then taking the mean of all points associated to one cluster as the new center. This will be repeated a constant number of iterations.

Ideas for Future Work:

- Distributed rank()/select() and succinct bit vectors for text search?
- Beyond `DIA<T>`? `Graph<V,E>`? `Matrix<T>`?
- Fault tolerance in the algorithms and scalability to large clusters.

Thank you for your attention!
Questions?